Wielokrotne wykorzystanie klas

Koncepcja ponownego wykorzystania klas polega na użyciu gotowych fragmentów oprogramowania (klas) do budowy innych klas. Gotowe klasy możemy wykorzystać na dwa sposoby:

- poprzez **kompozycję,**
- poprzez **dziedziczenie**.

✸ Kompozycja

Najprostszy sposób wykorzystania kodu klasy to bezpośrednie użycie obiektu tej klasy. Można jednak obiekt umieścić wewnątrz nowej klasy. Nazywamy to „tworzeniem obiektu składowego". Ponieważ tworzymy nowe klasy, używając klas już istniejących, dlatego koncepcja ta nazywa się kompozycją.

Z koncepcyjnego punktu widzenia kompozycja oznacza, że „obiekt jest zawarty w innym obiekcie".
Jest to relacja „całość – część" (B „zawiera" A) Relacja typu „posiada".

Składnia kompozycji jest prosta: wystarczy po prostu umieścić odwołania do obiektów wewnątrz nowo tworzonych klas.
*Przykładowo umieszczamy w naszej klasie kilka obiektów klasy **String**, kilka zmiennych podstawowych itp.*

```
class WaterSource
{
private String s;
WaterSource()
{
System.out.println("WaterSource()");
s = new String("Skonstruowany");
}
public String toString()
{
return s;
}
}

public class SprinklerSystem
{
private String valve1, valve2, valve3, valve4;
private WaterSource source;
private int i;
private float f;

public String toString()
{
return
"valve1 = " + valve1 + "\n" +
"valve2 = " + valve2 + "\n" +
"valve3 = " + valve3 + "\n" +
"valve4 = " + valve4 + "\n" +
"i = " + i + "\n" + "f = " + f + "\n" +
"źródło = " + source;
}
public static void main(String[] args)
{
SprinklerSystem sprinklers = new SprinklerSystem();
System.out.println(sprinklers);
}
}
```

Metoda **toString()** zdefiniowana w klasie **WaterSource** jest metodą specjalną. Jest ona wywoływana w momencie kiedy kompilator żąda łańcucha znaków. Przykładowo w miejscu:

```
System.out.println( "source = " + source);
```

Kompilator nie tworzy domyślnego obiektu dla każdej referencji. Aby odwołania były zainicjalizowane, należy zainicjalizować je:

- w miejscu definiowania obiektu,
- wewnątrz konstruktora danej klasy,
- tuż przed tym, jak zachodzi potrzeba użycia obiektu.

Dziedziczenie

Dziedziczenie jest jednym z podstawowych pojęć obiektowości i stanowi integralną część języka Java. Dziedziczenie używane jest nawet wtedy gdy nie dziedziczymy z innej klasy. Wszystkie klasy dziedziczą automatycznie klasę **Object**.

Przykład deklaracji dziedziczenia

```
class Róża extends Roślina    // dziedziczenie po klasie
class Samochód implements Pojazd, Towar // dziedziczenie po kilku interfesjach
class Fakty extends Prasa implements Codzienna, DoOglądania // dziedziczenie po klasie
i kilku interfejsach
```

Dziedziczenie polega na przejęciu własności i funkcjonalności innej klasy i ewentualnej ich modyfikacji w taki sposób, by były bardziej wyspecjalizowane.

Jest to relacja, nazywana generalizacją-specjalizacją B „jest typu" A, „B jest A", a jednocześnie B specjalizuje A. A jest generalizacją B.

Klasa dziedzicząca (pochodna) przejmuje wszystkie dopuszczone modyfikatorami dostępu składowe (“cechy”) klasy dziedziczonej. Aby zapisać w kodzie dziedziczenie należy użyć słowa kluczowego **extends**:

```
class Test extends KlasaBazowa
```

```
class Cleanser{
     private String s = new String("Cleanser");
     public void append(String a){
                                    s += a;
                                  }
      public void dilute() {
                        append(" dilute()");
                      }
       public void apply(){
                             append(" apply()");
                           }
        public void scrub(){
                            append(" scrub()");
                           }
        public String toString(){
                                 return s;
                                }
        public static void main(String[] args){
                                             Cleanser x = new Cleanser();
                                             x.dilute();
                                             x.apply();
                                             x.scrub();
                                    }
}
```

```
public class Detergent extends Cleanser{

public void scrub(){//Zmiana metody
                        append(" Detergent.scrub()");
                        super.scrub();//Wywołanie wersji z klasy bazowej
                    }

//Dodanie metod do interfejsu:
public void foam(){
                     append(" foam()");
                    }

//Testowanie nowej klasy:
public static void main(String[] args){
      Detergent x = new Detergent();
      x.dilute();
      x.apply();
      x.scrub();
      x.foam();
      System.out.println(x);
      System.out.println("Testowanie klasy bazowej:");
      Cleanser.main(args);
   }
}
```

Inicjalizacja klasy bazowej

W wyniku dziedziczenia otrzymaliśmy dwie klasy: bazową i jej pochodną. Nowa klasa posiada taki sam interfejs jak klasa bazowa, plus kilka dodatkowych pól i metod. Kiedy tworzymy obiekt klasy pochodnej, to zawiera ona w sobie klasę bazową jako podobiekt. Podobiekt klasy bazowej musi zostać poprawnie zainicjalizowany

```
class Art{
    Art(){
            System.out.println("Konstruktor klasy Art");
            }
      }
class Drawing extends Art{
    Drawing(){
            System.out.println("Konstruktor klasy Drawing");
            }
      }

public class Cartoon extends Drawing{
 public Cartoon(){
                  System.out.println("Konstruktor klasy Cartoon");
      }
      public static void main(String[] args){
      Cartoon x = new Cartoon();
      }
}
```

Wynik działania programu:

```
Konstruktor klasy Art
Konstruktor klasy Drawing
Konstruktor klasy Cartoon
```

Ale co zrobić w przypadku, gdy chcemy wywołać konstruktor z parametrami lub klasa nie posiada konstruktora domyślnego? Możemy zastosować słowo kluczowe super i wywołać odpowiedni konstruktor podając jego parametry.

```
class Game
{
Game(int i)
{
System.out.println("Konstruktor klasy Game");
}
}
class BoardGame extends Game
{
BoardGame(int i)
{
super(i);
System.out.println("Konstruktor klasy BoardGame");
}
}
```

```
public class Chess extends BoardGame
{
Chess()
{
super(11);
System.out.println("Konstruktor klasy Chess");
}
public static void main(String[] args)
{
Chess x = new Chess();
}
}
```

Wynik działania programu:

```
Konstruktor klasy Game
Konstruktor klasy BoardGame
Konstruktor klasy Chess
```